

Derived-term Automata for Extended Weighted Rational Expressions

Akim Demaille (2016-05-04 11:15:06 +0200 ab706b4)

LRDE, EPITA, akim@lrde.epita.fr

Contents

1	Introduction	2
2	Notations	3
2.1	Rational Series	3
2.2	Extended Weighted Rational Expressions	4
2.3	Rational Polynomials	5
2.4	Rational Expansions	6
2.5	Weighted Automata	7
3	Computing Expansions of Expressions	8
3.1	Expansion of a Rational Expression	8
3.2	Connection with Derivatives	9
4	Expansion-Based Derived-Term Automaton	9
4.1	Deterministic Automata	11
4.2	The Case of Complement	12
4.3	Complexity and Performances	12
5	Related Work	13
6	Conclusion	14
A	Appendix	15
B	Appendix: Proof of Theorem 20	16
B.1	Derivation by Words	16
B.2	Derived Terms	19
B.3	Derived-term Automaton	21

Abstract

We present an algorithm to build an automaton from a rational expression. This approach introduces support for extended weighted expressions. Inspired by derived-term based algorithms, its core relies on a different construct, *rational expansions*. We introduce an inductive algorithm to compute the expansion of an expression from which the automaton follows. This algorithm is independent of the size of the alphabet, and actually even supports infinite alphabets. It can easily be accommodated to generate deterministic (weighted) automata. These constructs are implemented in Vcsn, a free-software platform dedicated to weighted automata and rational expressions.



licensed under Creative Commons License CC-BY



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Foundational to Automata Theory, the Kleene Theorem (and its weighted extension, the Kleene–Schützenberger Theorem) states the equivalence of *recognizability* —accepted by an automaton— and *rationality* —defined by a *rational*, or *regular*, expression. Numerous constructive proofs (read *algorithms*) have been proposed to go from rational expressions to automata, and vice versa. This paper focuses on building an automaton from an expression.

In 1961 Glushkov [9] provides an algorithm to build a nondeterministic automaton (without spontaneous transitions) now often called the standard (or position, or Glushkov) automaton. Earlier (1960), McNaughton and Yamada [13] proposed the same construct for *extended* rational expressions (i.e., including intersection and complement operators), but performed the now usual subset-automaton construction on-the-fly, thus yielding a deterministic automaton. A key ingredient of these algorithms is that they build an automaton whose states represent positions in the rational expression, and computations on these automata actually represent “executions” of the rational expression.

Similarly, in 1964 Brzozowski [4] shows that *extended* expressions can be used directly as acceptors: transitions are “performed” by computing the left-quotient of the current expression by the current letter. With a proper equivalence relation between expressions (namely ACI: associativity, commutativity, and idempotence of the addition), Brzozowski shows that there is a finite number of equivalence classes of such quotients, called *derivatives*. This leads to a very natural construction of a *deterministic* automaton whose states are these derivatives. A rather discreet sentence (last line of p. 484) introduces the concept of “expansion”, which is not further developed.

In 1996 Antimirov [3] introduces a novel idea: do not apply ACI equivalence globally; rather, when computing the derivative of an expression which is a sum, split it in a set of “partial derivatives” (or “derived terms”) — which amounts to limiting ACI to the sums that are at the root of the expression. A key feature of the built automaton is that it is non-deterministic; as a result the worst-case size of resulting automaton is linear in the size of the expression, instead of exponential with Brzozowski’s construct. Antimirov also suggests *not* to rely on derivation in implementations, but on so called “linear forms”, which are closely related to Brzozowski’s expansions; derivation is used only to prove correctness.

In 2005 Lombardy and Sakarovitch [11] generalize the computation of the derivation and derived-term automaton to support weights. Since, as is well-known, not all weighted non deterministic automata can be determinized, their construct relies on a generalization of Antimirov’s derived-term that generates a *non-deterministic* automaton. In their formalization, Antimirov’s sets of derived terms naturally turn into *weighted* sets —each term is associated with a weight— that they name *polynomials* (of expressions). However, linear forms completely disappear, and the construction of the derived-term automaton relies on derivatives. Independently, and with completely different foundations, Rutten [15, 16] proposes a similar construction.

In 2011, Caron et al. [5] complete Antimirov’s construct to support extended expressions. This is at the price of a new definition of derivatives: sets of sets of expressions, interpreted as disjunctions of conjunctions of expressions.

The contributions of this paper are threefold. Firstly, we introduce “expansions”, which generalize Brzozowski’s expansions and Antimirov’s linear forms to support weighted expressions; they bind together the derivatives, the constant terms and the “firsts” of an expression. They make the computation of the derived-term automaton independent of the size of the alphabet, and actually completely eliminate the need for the alphabet to be finite.

Secondly, we provide support for extended weighted rational expressions, which generalizes both Lombardy and Sakarovitch [11] and Caron et al. [5]. And thirdly, we introduce a variation of this algorithm to build *deterministic* (weighted) automata.

We first settle the notations in Sect. 2, provide an algorithm to compute the expansion of an expression in Sect. 3, which is used in Sect. 4 to propose an alternative construction of the derived-term automaton. In Sect. 5 we expose related work and conclude in Sect. 6.

Interested readers may experiment with the concepts introduced here using Vcsn. Vcsn is a free-software platform dedicated to weighted automata and rational expressions [8]. It supports both derivations and expansions, as exposed in this paper, and the corresponding constructions of the derived-term automaton¹.

2 Notations

Our purpose is to define, compute, and use *rational expansions*. They intend to be to the differentiation (derivation) of rational expressions what differential forms are to the differentiation of functions. Defining expansions requires several concepts, defined bottom-up in this section. The following figure should help understanding these different entities, how they relate to each other, and where we are heading to: given a weighted rational expression $E_1 = \langle 5 \rangle 1 + \langle 2 \rangle ace + \langle 6 \rangle bce + \langle 4 \rangle ade + \langle 3 \rangle bde$ (weights are written in angle brackets), compute its expansion:

$$\begin{array}{c}
 \text{Weight} \quad \text{Letter} \quad \text{Expression (Sect. 2.2)} \quad \text{Monomial} \quad \text{Polynomial (Sect. 2.3)} \\
 \underbrace{\langle 5 \rangle}_{\text{Constant term}} \oplus \underbrace{a}_{\text{First}} \odot \underbrace{\left[\langle 2 \rangle \odot \underbrace{ce}_{\text{Derived term}} \right]}_{\text{Proper part of the expansion}} \oplus \underbrace{\left[\langle 4 \rangle \odot de \right]}_{\text{Proper part of the expansion}} \oplus b \odot \underbrace{\left[\langle 6 \rangle \odot ce \oplus \langle 3 \rangle \odot de \right]}_{\text{Proper part of the expansion}} \\
 \hline
 \text{Expansion (Sect. 2.4)}
 \end{array}$$

It is helpful to think of expansions as a normal form for expressions.

2.1 Rational Series

Series are to weighted automata what languages are to Boolean automata. Not all languages are rational (denoted by an expression), and similarly, not all series are rational (denoted by a weighted expression). We follow Sakarovitch [17].

Let A be a (finite) alphabet, and $\langle \mathbb{K}, +, \cdot, 0_{\mathbb{K}}, 1_{\mathbb{K}} \rangle$ a semiring whose (possibly non commutative) multiplication will be denoted by implicit concatenation. A (formal power) *series* over A^* with *weights* (or *multiplicities*) in \mathbb{K} is any map from A^* to \mathbb{K} . The weight of a word m in a series s is denoted $s(m)$. The *support* of a series s is the language of words that have a non-zero weight in s . The *empty* series, $m \mapsto 0_{\mathbb{K}}$, is denoted 0; for any word u (including ε), u denotes the series $m \mapsto 1_{\mathbb{K}}$ if $m = u$, $0_{\mathbb{K}}$ otherwise. Equipped with the pointwise addition ($s + t := m \mapsto s(m) + t(m)$) and the Cauchy product ($s \cdot t := m \mapsto \sum_{u,v \in A^* | u \cdot v = m} s(u) \cdot t(v)$) as multiplication, the set of these series forms a semiring denoted $\langle \mathbb{K} \langle A^* \rangle, +, \cdot, 0, \varepsilon \rangle$.

The *constant term* of a series s , denoted s_{ε} , is $s(\varepsilon)$, the weight of the empty word. A series s is *proper* if $s_{\varepsilon} = 0_{\mathbb{K}}$. The *proper part* of s , denoted s_p , is the proper series which coincides with s on non empty words: $s = s_{\varepsilon} + s_p$.

¹ See the interactive environment, <http://vcsn-sandbox.lrde.epita.fr>, or its documentation, http://vcsn.lrde.epita.fr/download/2.2/notebooks/expression.derived_term.html.

The *star* of a series is an infinite sum: $s^* := \sum_{n \in \mathbb{N}} s^n$. To ensure semantic soundness, we suppose that \mathbb{K} is a *topological semiring*, i.e., it is equipped with a topology, and both addition and multiplication are continuous. Besides, it is supposed to be *strong*, i.e., the product of two summable families is summable. This ensures that $\mathbb{K}\langle\langle A^* \rangle\rangle$, equipped with the product topology derived from the topology on \mathbb{K} , is also a strong topological semiring.

► **Proposition 1.** *Let \mathbb{K} be a strong topological semiring. Let $s \in \mathbb{K}\langle\langle A^* \rangle\rangle$, s^* is defined iff s_ε^* is defined and then $s^* = s_\varepsilon^* + s_\varepsilon^* s_p s^*$.*

Proof. By [17, Prop. 2.6, p. 396] s^* is defined iff s_ε^* is defined and then $s^* = (s_\varepsilon^* s_p)^* s_\varepsilon^* = s_\varepsilon^* (s_p s_\varepsilon^*)^*$. The result then follows directly from $s^* = \varepsilon + s s^*$: $s^* = s_\varepsilon^* (s_p s_\varepsilon^*)^* = s_\varepsilon^* (\varepsilon + (s_p s_\varepsilon^*) (s_p s_\varepsilon^*)^*) = s_\varepsilon^* + s_\varepsilon^* s_p (s_\varepsilon^* (s_p s_\varepsilon^*)^*) = s_\varepsilon^* + s_\varepsilon^* s_p s^*$. ◀

Rational languages are closed under intersection. Series support a natural generalization of intersection, the Hadamard product, which we name *conjunction* and denote $\&$. The conjunction of series s and t is defined as $s \& t := m \mapsto s(m) \cdot t(m)$.

Rational languages are also closed under complement, but generalizing this concept to series is more debatable. In the sequel, we will rely on the following definition: “ s^c is the characteristic series of the complement of the support of s .” More precisely, $s^c(m) := s(m)^c$ where $\forall k \in \mathbb{K}, k^c := 1_{\mathbb{K}}$ if $k = 0_{\mathbb{K}}$, $0_{\mathbb{K}}$ otherwise.

► **Proposition 2.** *For series $s, s', t, t', s_a, t_a \in \mathbb{K}\langle\langle A^* \rangle\rangle$ with $a \in A$, for $S, T \subseteq A$, and weights $k, h, s_\varepsilon, t_\varepsilon \in \mathbb{K}$:*

$$(s + s') \& t = s \& t + s' \& t \quad s \& (t + t') = s \& t + s \& t' \quad (ks) \& (ht) = (kh)(s \& t) \quad (1)$$

$$\left(s_\varepsilon + \sum_{a \in S} a \cdot s_a \right) \& \left(t_\varepsilon + \sum_{a \in T} a \cdot t_a \right) = s_\varepsilon t_\varepsilon + \sum_{a \in S \cap T} a \cdot (s_a \& t_a) \quad (2)$$

$$\left(s_\varepsilon + \sum_{a \in S} a \cdot s_a \right)^c = s_\varepsilon^c + \sum_{a \in S} a \cdot s_a^c + \sum_{a \in A \setminus S} a \cdot 0^c \quad (3)$$

2.2 Extended Weighted Rational Expressions

► **Definition 3** (Extended Weighted Rational Expression). *A rational (or regular) expression E is a term built from the following grammar, where $a \in A$ is a letter, and $k \in \mathbb{K}$ a weight: $E ::= 0 \mid 1 \mid a \mid E + E \mid \langle k \rangle E \mid E \langle k \rangle \mid E \cdot E \mid E^* \mid E \& E \mid E^c$.*

Since the product of \mathbb{K} does not need to be commutative there are two exterior products: $\langle k \rangle E$ and $E \langle k \rangle$. The *size* (aka *length*) of an expression E , $|E|$, is its number of symbols, excluding parentheses; its *width* (aka *literal length*), $\|E\|$, is the number of occurrences of letters.

Rational expressions are syntactic objects; they provide a finite notations for (some) series, which are semantic objects.

► **Definition 4** (Series Denoted by an Expression). *Let E be an expression. The series denoted by E , noted $\llbracket E \rrbracket$, is defined by induction on E :*

$$\begin{aligned} \llbracket 0 \rrbracket &:= 0 & \llbracket 1 \rrbracket &:= \varepsilon & \llbracket a \rrbracket &:= a & \llbracket E + F \rrbracket &:= \llbracket E \rrbracket + \llbracket F \rrbracket & \llbracket \langle k \rangle E \rrbracket &:= k \llbracket E \rrbracket \\ \llbracket E \langle k \rangle \rrbracket &:= \llbracket E \rrbracket k & \llbracket E \cdot F \rrbracket &:= \llbracket E \rrbracket \cdot \llbracket F \rrbracket & \llbracket E^* \rrbracket &:= \llbracket E \rrbracket^* & \llbracket E \& F \rrbracket &:= \llbracket E \rrbracket \& \llbracket F \rrbracket & \llbracket E^c \rrbracket &:= \llbracket E \rrbracket^c \end{aligned}$$

An expression is *valid* if it denotes a series. More specifically, this requires that $\llbracket F \rrbracket^*$ is well defined for each subexpression of the form F^* , i.e., that the constant term of $\llbracket F \rrbracket$ is *starrable*

in \mathbb{K} (Prop. 1). This definition, which involves series (semantics) to define a property of expressions (syntax), will be made effective (syntactic) with the appropriate definition of the constant term $c(E)$ of an expression E (Def. 16).

► **Example 5** ([11, Example 1]). Expressions $F_2 := \langle \frac{1}{6} \rangle a^* + \langle \frac{1}{3} \rangle b^*$, $E_2 = F_2^*$ have weights in \mathbb{Q} . F_2 is valid: its stars are on expressions that denote proper series. E_2 is valid, as the constant term of $\llbracket F_2 \rrbracket$ is $\frac{1}{6} + \frac{1}{3} = \frac{1}{2}$, whose star is defined: 2. $|E_2| = 8$, $\|E_2\| = 2$.

Two expressions E and F are *equivalent* iff $\llbracket E \rrbracket = \llbracket F \rrbracket$. Some expressions are “trivially equivalent”; any candidate expression will be rewritten via the following *trivial identities*. Any subexpression of a form listed to the left of a ‘ \Rightarrow ’ is rewritten as indicated on the right.

$$\begin{aligned}
E + 0 &\Rightarrow E & 0 + E &\Rightarrow E \\
\langle 0_{\mathbb{K}} \rangle E &\Rightarrow 0 & \langle 1_{\mathbb{K}} \rangle E &\Rightarrow E & \langle k \rangle 0 &\Rightarrow 0 & \langle k \rangle \langle h \rangle E &\Rightarrow \langle kh \rangle E \\
E \langle 0_{\mathbb{K}} \rangle &\Rightarrow 0 & E \langle 1_{\mathbb{K}} \rangle &\Rightarrow E & 0 \langle k \rangle &\Rightarrow 0 & E \langle k \rangle \langle h \rangle &\Rightarrow E \langle kh \rangle \\
\langle \langle k \rangle E \rangle \langle h \rangle &\Rightarrow \langle k \rangle \langle E \langle h \rangle \rangle & \ell \langle k \rangle &\Rightarrow \langle k \rangle \ell \\
E \cdot 0 &\Rightarrow 0 & 0 \cdot E &\Rightarrow 0 \\
\langle \langle k \rangle^? 1 \rangle \cdot E &\Rightarrow \langle k \rangle E & E \cdot \langle \langle k \rangle^? 1 \rangle &\Rightarrow E \langle k \rangle \\
0^* &\Rightarrow 1 \\
E \& 0 &\Rightarrow 0 & 0 \& E &\Rightarrow 0 & E \& 0^c &\Rightarrow E & 0^c \& E &\Rightarrow E \\
\langle k \rangle^? \ell \& \langle h \rangle^? \ell' &\Rightarrow \langle kh \rangle \ell & \langle k \rangle^? \ell \& \langle h \rangle^? \ell' &\Rightarrow 0 \\
\langle \langle k \rangle E \rangle^c &\Rightarrow E^c & (E \langle k \rangle)^c &\Rightarrow E^c
\end{aligned}$$

where E stands for a rational expression, $a \in A$ is a letter, $\ell, \ell' \in A \cup \{1\}$ denote two different labels, $k, h \in \mathbb{K}$ are weights, and $\langle k \rangle^? \ell$ denotes either $\langle k \rangle \ell$, or ℓ in which case $k = 1_{\mathbb{K}}$ in the right-hand side of \Rightarrow . The choice of these identities is beyond the scope of this paper (see [17]), however note that, with the exception of the last line, they are limited to trivial properties; in particular *linearity* (“weighted ACI”: associativity, commutativity, and $\langle k \rangle E + \langle h \rangle E \Rightarrow \langle k + h \rangle E$) is not enforced. In practice, additional identities help reducing the number of derived terms [14], hence the final automaton size. The last two rules, about complement, will be discussed in Sect. 4.2; they are disabled when \mathbb{K} has zero divisors.

► **Example 6.** Conjunction and complement can be combined to define new operators which are convenient syntactic sugar. For instance, $E \triangleleft F := E + (E^c \& F)$ allows to define a left-biased $+$ operator: $\llbracket E \triangleleft F \rrbracket(u) = \llbracket E \rrbracket(u)$ if $\llbracket E \rrbracket(u) \neq 0_{\mathbb{K}}$, $\llbracket F \rrbracket(u)$ otherwise. The following example mocks Lex-like scanners: identifiers are non-empty sequences of letters of $\{a, b\}$ that are not reserved keywords. The expression $E_3 := \langle 2 \rangle ab \triangleleft \langle 3 \rangle (a + b)^+$, with weights in \mathbb{Z} , maps the “keyword” ab to 2, and “identifiers” to 3. Once desugared and simplified by the trivial identities, we have $E_3 = \langle 2 \rangle ab + ((ab)^c \& \langle 3 \rangle ((a + b)(a + b)^*))$.

2.3 Rational Polynomials

At the core of the idea of “partial derivatives” introduced by Antimirov [3], is that of *sets* of rational expressions, later generalized in *weighted sets* by Lombardy and Sakarovitch [11], i.e., functions (partial, with finite domain) from the set of rational expressions into $\mathbb{K} \setminus \{0_{\mathbb{K}}\}$. It proves useful to view such structures as “polynomials of rational expressions”. In essence, they capture the linearity of addition.

► **Definition 7** (Rational Polynomial). A polynomial (of rational expressions) is a finite (left) linear combination of rational expressions. Syntactically it is represented by a term built from the grammar $P ::= 0 \mid \langle k_1 \rangle \odot E_1 \oplus \cdots \oplus \langle k_n \rangle \odot E_n$ where $k_i \in \mathbb{K} \setminus \{0_{\mathbb{K}}\}$ denote non-null weights, and E_i denote non-null expressions. Expressions may not appear more than once in a polynomial. A monomial is a pair $\langle k_i \rangle \odot E_i$.

We use specific symbols (\odot and \oplus) to clearly separate the outer polynomial layer from the inner expression layer. A polynomial P of rational expressions can be “projected” as a rational expression $\text{expr}(P)$ by mapping its sum and left-multiplication by a weight onto the corresponding operators on rational expressions. This operation is performed on a canonical form of the polynomial (expressions are sorted in a well defined order). Polynomials denote series: $\llbracket P \rrbracket := \llbracket \text{expr}(P) \rrbracket$.

► **Example 8.** Let $E_1 := \langle 5 \rangle 1 + \langle 2 \rangle ace + \langle 6 \rangle bce + \langle 4 \rangle ade + \langle 3 \rangle bde$. Polynomial $P_{1a} := \langle 2 \rangle \odot ce \oplus \langle 4 \rangle \odot de$ has two monomials: $\langle 2 \rangle \odot ce$ and $\langle 4 \rangle \odot de$. It denotes the (left) quotient of $\llbracket E_1 \rrbracket$ by a , and $P_{1b} := \langle 6 \rangle \odot ce \oplus \langle 3 \rangle \odot de$ the quotient by b .

Let $P = \langle k_1 \rangle \odot E_1 \oplus \cdots \oplus \langle k_n \rangle \odot E_n$ be a polynomial, k a weight (possibly null) and F an expression (possibly null), we introduce the following operations:

$$\begin{aligned} P \cdot F &:= \langle k_1 \rangle \odot (E_1 \cdot F) \oplus \cdots \oplus \langle k_n \rangle \odot (E_n \cdot F) \\ \langle k \rangle P &:= \langle k k_1 \rangle \odot E_1 \oplus \cdots \oplus \langle k k_n \rangle \odot E_n & P \langle k \rangle &:= \langle k_1 \rangle \odot (E_1 \langle k \rangle) \oplus \cdots \oplus \langle k_n \rangle \odot (E_n \langle k \rangle) \\ P_1 \& P_2 &:= \bigoplus_{\substack{\langle k_1 \rangle \odot E_1 \in P_1 \\ \langle k_2 \rangle \odot E_2 \in P_2}} \langle k_1 k_2 \rangle \odot (E_1 \& E_2) & P^c &:= \langle 1_{\mathbb{K}} \rangle \odot \text{expr}(P)^c \end{aligned} \quad (4)$$

Trivial identities might simplify the result, e.g., $(\langle 1_{\mathbb{K}} \rangle \odot a) \& (\langle 1_{\mathbb{K}} \rangle \odot b) = \langle 1_{\mathbb{K}} \rangle \odot (a \& b) = 0$.

Note the asymmetry between left and right exterior products. The addition of polynomials is commutative, multiplication by zero (be it an expression or a weight) evaluates to the null polynomial, and the left-multiplication by a weight is distributive.

► **Lemma 9.** $\llbracket P \cdot F \rrbracket = \llbracket P \rrbracket \cdot \llbracket F \rrbracket \quad \llbracket \langle k \rangle P \rrbracket = \langle k \rangle \llbracket P \rrbracket \quad \llbracket P \langle k \rangle \rrbracket = \llbracket P \rrbracket \langle k \rangle$
 $\llbracket P_1 \& P_2 \rrbracket = \llbracket P_1 \rrbracket \& \llbracket P_2 \rrbracket \quad \llbracket P^c \rrbracket = \llbracket P \rrbracket^c$.

Proof. The first three are trivial. The case of $\&$ follows from (1). Complement follows from its definition: $\llbracket P^c \rrbracket := \llbracket \text{expr}(P^c) \rrbracket = \llbracket \langle 1_{\mathbb{K}} \rangle \odot \text{expr}(P)^c \rrbracket = \llbracket \text{expr}(P)^c \rrbracket = \llbracket \text{expr}(P) \rrbracket^c = \llbracket P \rrbracket^c$. ◀

2.4 Rational Expansions

► **Definition 10** (Rational Expansion). A rational expansion X is a term built from the grammar $X ::= \langle k \rangle \oplus a_1 \odot [P_1] \oplus \cdots \oplus a_n \odot [P_n]$ where $k \in \mathbb{K}$ is a weight (possibly null), $a_i \in A$ letters (occurring at most once), and P_i non-null polynomials. We name k the constant term, $a_1 \odot [P_1] \oplus \cdots \oplus a_n \odot [P_n]$ the proper part, and $\{a_1, \dots, a_n\}$ (possibly empty) the firsts.

To ease reading, polynomials are written in square brackets. Contrary to expressions and polynomials, there is no specific term for the empty expansion: it is represented by $\langle 0_{\mathbb{K}} \rangle$, the null weight. Except for this case, null constant terms are left implicit. Besides their support for weights, expansions differ from Antimirov’s linear forms in that they integrate the constant term, which gives them a flavor of series. Given an expansion X , we denote by X_ϵ (or $X(\epsilon)$) its constant term, by $f(X)$ its firsts, by X_p its proper part, and by X_a (or $X(a)$) the polynomial corresponding to a in X . Expansions will thus be written: $X = \langle X_\epsilon \rangle \oplus \bigoplus_{a \in f(X)} a \odot [X_a]$.

An expansion whose polynomials are monomials is said to be *deterministic*. An expansion X can be “projected” as a rational expression $\text{expr}(X)$ by mapping weights, letters and polynomials to their corresponding rational expressions, and \oplus/\odot to the sum/concateration of rational expressions. Again, this is performed on a canonical form of the expansion: letters and polynomials are sorted. Expansions also denote series: $\llbracket X \rrbracket := \llbracket \text{expr}(X) \rrbracket$. An expansion X is said to be *equivalent* to an expression E iff $\llbracket X \rrbracket = \llbracket E \rrbracket$.

► **Example 11** (Ex. 8 continued). *Expansion $X_1 := \langle 5 \rangle \oplus a \odot [P_{1a}] \oplus b \odot [P_{1b}]$ has $X_1(\varepsilon) = \langle 5 \rangle$ as constant term, and maps the letter a (resp. b) to the polynomial $X_1(a) = P_{1a}$ (resp. $X_1(b) = P_{1b}$). X_1 can be proved to be equivalent to E_1 .*

Let X, Y be expansions, k a weight, and E an expression (all possibly null):

$$X \oplus Y := \langle X_\varepsilon + Y_\varepsilon \rangle \oplus \bigoplus_{a \in f(X) \cup f(Y)} a \odot [X_a \oplus Y_a] \quad (5)$$

$$\langle k \rangle X := \langle k X_\varepsilon \rangle \oplus \bigoplus_{a \in f(X)} a \odot [\langle k \rangle X_a] \quad X \langle k \rangle := \langle X_\varepsilon k \rangle \oplus \bigoplus_{a \in f(X)} a \odot [X_a \langle k \rangle] \quad (6)$$

$$X \cdot E := \bigoplus_{a \in f(X)} a \odot [X_a \cdot E] \quad \text{with } X \text{ proper: } X_\varepsilon = 0_{\mathbb{K}} \quad (7)$$

$$X \& Y := \langle X_\varepsilon Y_\varepsilon \rangle \oplus \bigoplus_{a \in f(X) \cap f(Y)} a \odot [X_a \& Y_a] \quad (8)$$

$$X^c := \langle X_\varepsilon^c \rangle \oplus \bigoplus_{a \in f(X)} a \odot [X_a^c] \oplus \bigoplus_{a \in A \setminus f(X)} a \odot [0^c] \quad (9)$$

Since by definition expansions never map to null polynomials, some firsts might be smaller that suggested by these equations. For instance in \mathbb{Z} the sum of $\langle 1 \rangle \oplus a \odot [\langle 1 \rangle \odot b]$ and $\langle 1 \rangle \oplus a \odot [\langle -1 \rangle \odot b]$ is $\langle 2 \rangle$, and $(a \odot [\langle 1 \rangle \odot b]) \& (a \odot [\langle 1 \rangle \odot c])$ is $\langle 0 \rangle$ since $b \& c \Rightarrow 0$. Note that X^c is a deterministic expansion.

The following lemma is simple to establish: lift semantic equivalences, such as those of Prop. 2, to syntax, using Lemma 9.

► **Lemma 12.** $\llbracket X \oplus Y \rrbracket = \llbracket X \rrbracket + \llbracket Y \rrbracket \quad \llbracket \langle k \rangle X \rrbracket = \langle k \rangle \llbracket X \rrbracket \quad \llbracket X \langle k \rangle \rrbracket = \llbracket X \rrbracket \langle k \rangle$
 $\llbracket X \cdot E \rrbracket = \llbracket X \rrbracket \cdot \llbracket E \rrbracket \quad \llbracket X \& Y \rrbracket = \llbracket X \rrbracket \& \llbracket Y \rrbracket \quad \llbracket X^c \rrbracket = \llbracket X \rrbracket^c.$

2.5 Weighted Automata

► **Definition 13** (Automaton). *A weighted automaton \mathcal{A} is a tuple $\langle A, \mathbb{K}, Q, E, I, T \rangle$ where:*

- A (the set of labels) is an alphabet (usually finite),
- \mathbb{K} (the set of weights) is a semiring,
- Q is a set of states,
- I and T are the initial and final functions from Q into \mathbb{K} ,
- E is a (partial) function from $Q \times A \times Q$ into $\mathbb{K} \setminus \{0_{\mathbb{K}}\}$;
its domain represents the transitions: (source, label, destination).

An automaton is *locally finite* if each state has a finite number of outgoing transitions ($\forall s \in Q, \{s\} \times A \times Q \cap E$ is finite). A *finite automaton* has a finite number of states. A *path* p in an automaton is a sequence of transitions $(q_0, a_0, q_1)(q_1, a_1, q_2) \cdots (q_n, a_n, q_{n+1})$ where the source of each is the destination of the previous one; its *label* is the word $a_0 a_1 \cdots a_n$, its *weight* is $I(q_0) \otimes E(q_0, a_0, q_1) \otimes \cdots \otimes E(q_n, a_n, q_{n+1}) \otimes T(q_{n+1})$. The *evaluation* of word u by a locally finite automaton \mathcal{A} , $\mathcal{A}(u)$, is the (finite) sum of the weights of all the paths

labeled by u , or $0_{\mathbb{K}}$ if there are no such path. The *behavior* of such an automaton \mathcal{A} is the series $\llbracket \mathcal{A} \rrbracket := u \mapsto \mathcal{A}(u)$. A state q is *initial* if $I(q) \neq 0_{\mathbb{K}}$. A state q is *accessible* if there is a path from an initial state to q . The *accessible* part of an automaton \mathcal{A} is the subautomaton whose states are the accessible states of \mathcal{A} . The size of a finite automaton, $|\mathcal{A}|$, is its number of states.

We are interested, given an expression E , by an algorithm to compute an automaton \mathcal{A}_E such that $\llbracket \mathcal{A}_E \rrbracket = \llbracket E \rrbracket$ (Sect. 4). To this end, we first introduce a simple recursive procedure to compute *the* expansion of an expression.

3 Computing Expansions of Expressions

3.1 Expansion of a Rational Expression

► **Definition 14** (Expansion of a Rational Expression). *The expansion of a rational expression E , written $d(E)$, is the expansion defined inductively as follows:*

$$d(0) := \langle 0_{\mathbb{K}} \rangle \quad d(1) := \langle 1_{\mathbb{K}} \rangle \quad d(a) := a \odot [\langle 1_{\mathbb{K}} \rangle \odot 1] \quad (10)$$

$$d(E + F) := d(E) \oplus d(F) \quad d(\langle k \rangle E) := \langle k \rangle d(E) \quad d(E \langle k \rangle) := d(E) \langle k \rangle \quad (11)$$

$$d(E \cdot F) := d_p(E) \cdot F \oplus \langle d_\varepsilon(E) \rangle d(F) \quad (12)$$

$$d(E^*) := \langle d_\varepsilon(E)^* \rangle \oplus \langle d_\varepsilon(E)^* \rangle d_p(E) \cdot E^* \quad (13)$$

$$d(E \& F) := d(E) \& d(F) \quad (14)$$

$$d(E^c) := d(E)^c \quad (15)$$

where $d_\varepsilon(E) := d(E)_\varepsilon$, $d_p(E) := d(E)_p$ are the constant term/proper part of $d(E)$.

The right-hand sides are indeed expansions. The computation trivially terminates: induction is performed on strictly smaller subexpressions. These formulas are enough to compute the expansion of an expression; there is no secondary process for the firsts — indeed $d(a) := a \odot [\langle 1_{\mathbb{K}} \rangle \odot 1]$ suffices and every other case simply propagates or assembles the firsts — or the constant terms. Of course, in an implementation, a single recursive call to $d(E)$ is performed for (12) and (13), from which $d_\varepsilon(E)$ and $d_p(E)$ are obtained. So for instance (13) should rather be written: $d(E^*) := \text{let } X = d(E) \text{ in } \langle X^* \rangle \oplus \langle X^* \rangle X_p \cdot E^*$. Besides, existing expressions should be referenced to, not duplicated: in the previous piece of code, E^* is not built again, the input argument is reused.

► **Proposition 15.** *The expansion of a rational expression is equivalent to the expression.*

Proof. We prove that $\llbracket d(E) \rrbracket = \llbracket E \rrbracket$ by induction on the expression. The equivalence is straightforward for (10) and (11). The case of multiplication, (12), follows from:

$$\begin{aligned} \llbracket d(E \cdot F) \rrbracket &= \llbracket d_p(E) \cdot F \oplus \langle d_\varepsilon(E) \rangle d(F) \rrbracket = \llbracket d_p(E) \rrbracket \cdot \llbracket F \rrbracket + \langle \llbracket d_\varepsilon(E) \rrbracket \rangle \cdot \llbracket d(F) \rrbracket \\ &= \llbracket d_p(E) \rrbracket \cdot \llbracket F \rrbracket + \langle \llbracket d_\varepsilon(E) \rrbracket \rangle \cdot \llbracket F \rrbracket = \left(\llbracket \langle d_\varepsilon(E) \rangle \rrbracket + \llbracket d_p(E) \rrbracket \right) \cdot \llbracket F \rrbracket \\ &= \llbracket \langle d_\varepsilon(E) \rangle + d_p(E) \rrbracket \cdot \llbracket F \rrbracket = \llbracket d(E) \rrbracket \cdot \llbracket F \rrbracket = \llbracket E \rrbracket \cdot \llbracket F \rrbracket = \llbracket E \cdot F \rrbracket \end{aligned}$$

It might seem more natural to exchange the two terms (i.e., $\langle d_\varepsilon(E) \rangle \cdot d(F) \oplus d_p(E) \cdot F$), but an implementation first computes $d(E)$ and then computes $d(F)$ *only if* $d_\varepsilon(E) \neq 0_{\mathbb{K}}$. The case of Kleene star, (13), follows from Prop. 1. The case of conjunction is straightforward:

$$\llbracket d(E \& F) \rrbracket = \llbracket d(E) \& d(F) \rrbracket \quad \text{by definition, (14)}$$

$$\begin{aligned}
&= \llbracket d(E) \rrbracket \& \llbracket d(F) \rrbracket && \text{by Lemma 12} \\
&= \llbracket E \rrbracket \& \llbracket F \rrbracket && \text{by induction hypothesis} \\
&= \llbracket E \& F \rrbracket && \text{by Lemma 12} \quad \blacktriangleleft
\end{aligned}$$

$$\llbracket d(E \& F) \rrbracket = \llbracket d(E) \& d(F) \rrbracket = \llbracket d(E) \rrbracket \& \llbracket d(F) \rrbracket = \llbracket E \rrbracket \& \llbracket F \rrbracket = \llbracket E \& F \rrbracket$$

3.2 Connection with Derivatives

We reproduce here the definition of constant terms and derivatives from Lombardy et al [11, p. 148 and Def. 2], with our notations and added support for extended expressions.

► **Definition 16** (Constant Term and Derivative).

$$c(0) := \langle 0_{\mathbb{K}} \rangle, \quad c(1) := \langle 1_{\mathbb{K}} \rangle, \quad \partial_a 0 := 0, \quad \partial_a 1 := 0, \quad (16)$$

$$c(a) := \langle 0_{\mathbb{K}} \rangle, \forall a \in A, \quad \partial_a b := 1 \text{ if } b = a, 0 \text{ otherwise}, \quad (17)$$

$$c(E + F) := c(E) + c(F), \quad \partial_a (E + F) := \partial_a E \oplus \partial_a F, \quad (18)$$

$$c(\langle k \rangle E) := \langle k \rangle c(E), \quad \partial_a (\langle k \rangle E) := \langle k \rangle (\partial_a E), \quad (19)$$

$$c(E \langle k \rangle) := c(E) \langle k \rangle, \quad \partial_a (E \langle k \rangle) := (\partial_a E) \langle k \rangle, \quad (20)$$

$$c(E \cdot F) := c(E) \cdot c(F), \quad \partial_a (E \cdot F) := (\partial_a E) \cdot F \oplus \langle c(E) \rangle \partial_a F, \quad (21)$$

$$c(E^*) := c(E)^*, \quad \partial_a E^* := \langle c(E)^* \rangle (\partial_a E) \cdot E^* \quad (22)$$

$$c(E \& F) := c(E) \cdot c(F), \quad \partial_a (E \& F) := \partial_a E \& \partial_a F, \quad (23)$$

$$c(E^c) := c(E)^c, \quad \partial_a E^c := (\partial_a E)^c \quad (24)$$

where (22) applies iff $c(E)^*$ is defined in \mathbb{K} .

The reader is invited to compare Def. 14 and Def. 16, which does not even include the computation of the firsts.

► **Proposition 17.** *For any rational expression E , $d(E)(\varepsilon) = c(E)$, and $d(E)(a) = \partial_a E$.*

Proof. A straightforward induction on E . The cases of constants and letters are immediate consequences of (16) and (17) on the one hand, and (10) on the other hand. (11) and (18) both express straightforward “linearity”. Multiplication (concatenation) is again barely a change of notation between (12) and (21), and likewise for the Kleene star ((13) and (22)). Conjunction, (23), follows from (8) and (14), and complement, (24), from (15) and (9). ◀

Prop. 17 states that expansions, like Antimirov’s linear forms, offer a different means to compute the expression derivatives. However expansions seem to better capture the essence of the process, where the computations of constant terms are tightly coupled with that of the derivations. The formulas are more concise. Expansions are also “more complete” than derivations, viz., the expansion of an expression can be seen as a normal-form of this expression: $E \equiv \text{expr}(d(E))$ and $d(E) = d(\text{expr}(d(E)))$. Expansions are more efficient to perform effective calculations, such building an automaton (Sect. 4.3), while derivatives are used to prove the correctness (Theorem 20).

4 Expansion-Based Derived-Term Automaton

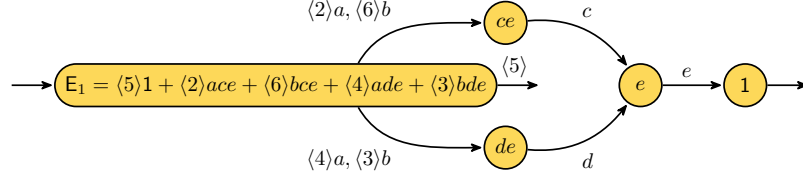
► **Definition 18** (Derived-Term Automaton). *The derived-term automaton of an expression E is the accessible part of the automaton $\mathcal{A}_E := \langle A, \mathbb{K}, Q, E, I, T \rangle$ defined as follows:*

- Q is the set of rational expressions on alphabet A with weights in \mathbb{K} ,
- $E(F, a, F') = k$ iff $a \in f(d(F))$ and $\langle k \rangle F' \in d(F)(a)$,
- $I = E \mapsto 1_{\mathbb{K}}$, $T(F) = k$ iff $\langle k \rangle = d(F)(\varepsilon)$.

The resulting automaton is locally finite, and not necessarily deterministic: given a state F and $a \in f(d(F))$ one of its firsts, the “destinations” are all the expressions of $d(F)(a)$.

► **Example 19** (Ex. 8 and 11 continued). Given $d(E_1)$, \mathcal{A}_{E_1} follows.

$$d(E_1) = X_1 = \langle 2 \rangle \oplus a \odot [\langle 2 \rangle \odot ce \oplus \langle 4 \rangle \odot de] \oplus b \odot [\langle 6 \rangle \odot ce \oplus \langle 3 \rangle \odot de]$$



It is straightforward to extract an algorithm from Def. 18, using a work-list of states whose outgoing transitions to compute. This approach admits a natural lazy implementation: the whole automaton is not computed at once, but rather, states and transitions are computed on-the-fly, on demand, for instance when evaluating a word.

► **Theorem 20.** Any (valid) expression E and its expansion-based derived-term automaton \mathcal{A}_E denote the same series, i.e., $\llbracket \mathcal{A}_E \rrbracket = \llbracket E \rrbracket$.

The smallness of the derived-term automaton for basic operators ($|\mathcal{A}_E| \leq \|E\| + 1$ [11, Theorem 2]) no longer applies with extended operators. Let m and n be coprime integers, $E := (a^m)^* \& (a^n)^*$ has width $\|E\| = m+n$; it is easy to see that $|\mathcal{A}_E| = mn$. It is also a classical result that the minimal (trim) automaton to recognize the language of $F_n := (a+b)^* a(a+b)^n$ has 2^{n+1} states; so $\|F_n^c\| = 2n+3$, but $|\mathcal{A}_{F_n^c}| = 2^{n+1} + 1$ (the additional state is the sink state needed to get a *complete* deterministic automaton before complement). Actually, when complement is used on infinite semiring, it is not even guaranteed that the automaton is finite (Sect. 4.2).

Sketch of proof of Theorem 20, see Appendix B. This result is proved as [11, Theorem 4]: it requires several lemmas whose proofs are simple, but long.

First define the derivation with respect to a word as the repetition of derivation with respect to a letter, and prove that $\llbracket \partial_u E \rrbracket = u^{-1} \llbracket E \rrbracket$.

Second, prove that the set of derivatives of an expression E with respect to words is generated by $D(E)$, a set of expressions, called *derived terms*. The states of the derived-term automaton are not any expressions, they are derived terms (and E itself), so the finiteness of $D(E)$ implies that of the automaton.

$D(E)$ admits a simple inductive computation [11, Definition 3], to which we add:

$$\begin{aligned} D(E \& F) &:= \{E_i \& F_j \mid \forall E_i \in D(E), \forall F_j \in D(F)\} \\ D(E^c) &:= \{(\langle k_1 \rangle E_1 + \dots + \langle k_n \rangle E_n)^c \mid \forall k_1, \dots, k_n \in \mathbb{K}, \forall E_1, \dots, E_n \in D(E)\} \end{aligned} \quad (25)$$

If E features no complement, $D(E)$ is trivially finite. Equation (25) is related to *determinized* expansions (Sect. 4.1): in essence it dubs (complements of) all potential derivatives of E into derived-terms (comparable to going from Antimirov’s partial derivatives to Brzozowski’s derivatives). On infinite semirings, $D(E^c)$ is infinite (more about this in Sect. 4.2). However, on finite semirings, such as \mathbb{B} , it is finite, albeit potentially large.

Finally, prove that $\llbracket \mathcal{A}_E \rrbracket(u) = \llbracket E \rrbracket(u)$ for all words $u \in A^*$. ◀

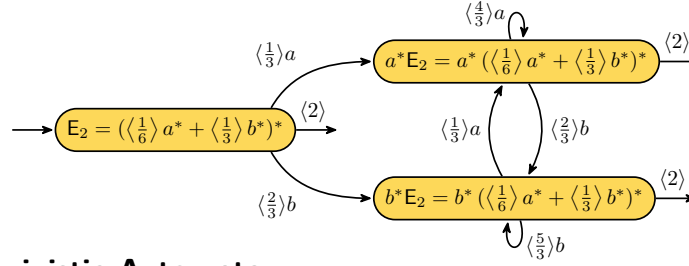
► **Example 21** (Ex. 5 continued). To compute the expansion of E_2 , one has:

$$\begin{aligned} d(F_2) &= \langle \tfrac{1}{2} \rangle \oplus a \odot \left[\langle \tfrac{1}{6} \rangle \odot a^* \right] \oplus b \odot \left[\langle \tfrac{1}{3} \rangle \odot b^* \right] \\ d(\boxed{E_2}) &= d(F_2^*) = \langle d_\varepsilon(F_2)^* \rangle \oplus \langle d_\varepsilon(F_2)^* \rangle d_p(F_2) \cdot F_2^* \\ &= \langle 2 \rangle \oplus a \odot \left[\langle \tfrac{1}{3} \rangle \odot \boxed{a^* E_2} \right] \oplus b \odot \left[\langle \tfrac{2}{3} \rangle \odot \boxed{b^* E_2} \right] \end{aligned}$$

The derived terms of E_2 are E_2 , $a^* E_2$, and $b^* E_2$:

$$\begin{aligned} d(\boxed{a^* E_2}) &= \langle 2 \rangle \oplus a \odot \left[\langle \tfrac{4}{3} \rangle \odot \boxed{a^* E_2} \right] \oplus b \odot \left[\langle \tfrac{2}{3} \rangle \odot \boxed{b^* E_2} \right] \\ d(\boxed{b^* E_2}) &= \langle 2 \rangle \oplus a \odot \left[\langle \tfrac{1}{3} \rangle \odot \boxed{a^* E_2} \right] \oplus b \odot \left[\langle \tfrac{5}{3} \rangle \odot \boxed{b^* E_2} \right] \end{aligned}$$

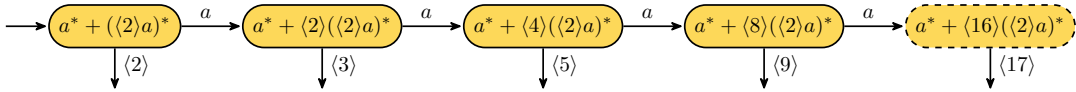
The derived-term automaton of E_2 is therefore:



4.1 Deterministic Automata

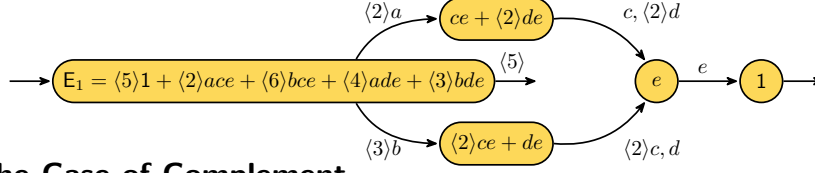
The exposed approach can be used to generate *deterministic* automata by *determinizing* the expansions: $\det(X) := \bigoplus_{a \in f(X)} \langle 1_{\mathbb{K}} \rangle \odot \text{expr}(X_a)$. The expr operator “consolidates” a polynomial into an expression that ensures this determinism. For instance the expansion $a \odot [\langle 1_{\mathbb{K}} \rangle \odot b \oplus \langle 1_{\mathbb{K}} \rangle \odot c]$, which would yield two transitions labeled by a , one to b and the other to c , is determinized into $a \odot [\langle 1_{\mathbb{K}} \rangle \odot (b + c)]$, yielding a single transition, to $b + c$.

It is well known that some nondeterministic *weighted* automata have no deterministic equivalent, in which case determinization loops. Our construct is subject to the same condition. The expression $E := a^* + (\langle 2 \rangle a)^*$ on the alphabet $\{a\}$ admits an infinite number of derivatives: $\partial_{a^n}(E) = a^* \oplus \langle 2^n \rangle (\langle 2 \rangle a)^*$. Therefore our construction of *deterministic* automata would not terminate: the automaton is locally finite but infinite (and there is no finite deterministic automaton equivalent to E). However, a lazy implementation as available in Vcsn¹ would uncover the automaton on demand, for instance when evaluating a word.



To improve determinizability, when \mathbb{K} features a left-division, we apply the usual technique used in weighted determinization implementations: normalize the results to keep a unique representative of colinear polynomials. Concretely, when determinizing expansions, polynomials are first normalized: $\det(X) := \bigoplus_{a \in f(X)} \langle |X_a| \rangle \odot \text{expr}(|X_a| \setminus X_a)$ where, for a polynomial $P = \bigoplus_{i \in I} \langle k_i \rangle \odot E_i$, and a weight k , $k \setminus P := \bigoplus_{i \in I} \langle k \setminus k_i \rangle \odot E_i$, and the weight $|P|$ denotes some “norm” of (the coefficients of) P . For instance $|P|$ can be the GCD of the k_i (so that the coefficients are coprime), or, in the case of a field, the first non null k_i (so that the first non null coefficient is $1_{\mathbb{K}}$), or the sum of the k_i provided it’s not null (so that the sum of the coefficients is $1_{\mathbb{K}}$), etc.

► **Example 22** (Ex. 8, 11 and 19 cont.). The deterministic derived-term automaton of E_1 using GCD-normalization is:



4.2 The Case of Complement

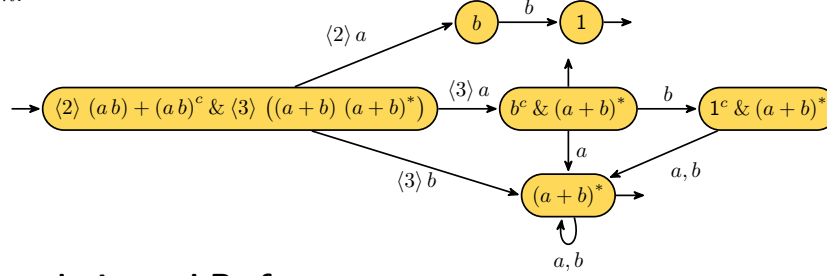
It is well known that to complement an (unweighted) automaton, it needs to be deterministic and complete (which can lead to an exponential number of states). “Local” determinism (i.e., restricted to complemented subexpressions) is ensured by **expr** in the definition of the complement of an expansion in (4) and (9).

In the case of weighted expressions, we hit the same problems—and apply the same techniques—as in Sect. 4.1: not all expressions generate finite automata. A strict (non-lazy) implementation would not terminate on $(a^* + (\langle 2 \rangle a)^*)^c$; a lazy implementation would uncover finite portions of the automaton, on demand. However, although $F := (\langle 2 \rangle a)^* + (\langle 4 \rangle aa)^*$ admits an infinite number of derivatives, F^c features only two: $(\langle 2 \rangle (\langle 2 \rangle a)^* + \langle 4 \rangle (a(\langle 4 \rangle aa)^*))^c \Rightarrow ((\langle 2 \rangle a)^* + \langle 2 \rangle (a(\langle 4 \rangle aa)^*))^c$ and itself. It is the trivial identity $(\langle k \rangle E)^c \Rightarrow E^c$ that eliminates the common factor.

► **Example 23** (Ex. 6 continued). We have (see Ex. 24 in Appendix A for details):

$$d(E_3) = a \odot [\langle 2 \rangle \odot b \oplus \langle 3 \rangle \odot (b^c \& (a+b)^*)] \oplus b \odot [\langle 3 \rangle \odot (a+b)^*]$$

The lower part of \mathcal{A}_{E_3} is characteristic of the complement of a complete deterministic automaton:



4.3 Complexity and Performances

We focus on basic expressions. Obviously, $\|E\| \leq |E|$, and we know $|\mathcal{A}_E| \leq \|E\| + 1$.

The complexity of Antimirov’s algorithm is $O(\|E\|^3 |E|^2)$ [6]: for each of the $|\mathcal{A}_E|$ states, we may generate at most $|\mathcal{A}_E|$ partial derivatives, each one to compare to the $|\mathcal{A}_E|$ derived-terms. That’s $O(|\mathcal{A}_E|^3)$ comparisons to perform on objects of size $O(|E|^2)$.

However, hash tables allow to avoid these costly comparisons. For each of the $|\mathcal{A}_E|$ states, we may generate at most $|\mathcal{A}_E|$ partial derivatives and number them via a hash table. Computing an expansion builds an object of size $O(|E|^2)$, however using references instead of deep copies allows to stay linear, so the complexity is $O(\|E\|^2 |E|)$.

To build the derived-term automaton using derivation, one loops over the alphabet for each derived term. This incurs a performance penalty with large alphabets. The following table reports the duration of the process, in milliseconds, for $E_n := (a+b)^* a (a+b)^n$ (right associative) by Vcsn², depending on n , for two alphabet sizes: 2 and 254 (Vcsn reserves two chars).

² Vcsn 2.2 as of 2016-01-29, compiled with Clang 3.6 with options `-O3 -DNDEBUG`, and run on a Mac OS X 10.11.3, Intel Core i7 2.9GHz, 8GB of RAM. Best run out of five.

	5	10	50	100	500	1000	5000
derivation 2	0.08	0.12	0.80	2.5	55	210	4,735
derivation 254	1.12	2.15	15.56	39.2	694	2,448	59,019
expansion 2	0.08	0.10	0.55	1.2	20	70	1,617
expansion 254	0.08	0.11	0.49	1.2	19	70	1,619

Even on a two-letter alphabet, the expansion-based algorithm performs better than the derivation-based one. (To put things in perspective, the construction of the standard automaton for $n = 5000$ takes 8.2ms.)

One can optimize the derivation-based algorithm by computing the firsts globally [14] or locally, on-the-fly, and then derivating on this set. However, on sums such as $a_1 + \dots + a_n$ (where a_i are distinct letters) the expansion requires a single traversal ($O(n)$) whereas one still needs n derivations, a $O(n^2)$ process. Besides, the derivation-based algorithm computes the constant term of an expression several times: to check whether the current state is final, to compute the derivation of products and stars, and to compute the firsts of products. To fix this issue, these repeated computations can be cached.

Addressing both concerns (iteration over the alphabet, repeated computation of the constant term) for the derivation-based algorithm requires three tightly entangled algorithms (constant term, derivation, first). Expansions, on the other hand, keep them together, in a single construct, computed in a single traversal of the expression.

5 Related Work

Compared to Brzozowski [4] we introduced *weighted* expansions, and their direct computation, making them the core computation of the algorithm. This was partly done for basic Boolean expressions by Antimirov [3] as “linear forms”.

Aside from our support for weighted expressions, our approach of extended operators is comparable to that of Caron et al. [5], but, we believe, using a simpler framework. Basically, their sets of sets of expressions correspond to polynomials of conjunctions: their $\{\{E, F\}, \{G, H\}\}$ is our $E \& F \oplus G \& H$. Using our framework, the automaton of Fig. 3 [5] has one state less, since $\{E, F\}$ and $\{E \cap F\}$ both are $E \& F$. Actually, the main point of sets of sets of expressions is captured by our *distributive* definition of the conjunction of polynomials, (4), which matches that of their \odot operator; indeed what they call the “natural extension” [5, Sect. 3.1] would correspond to $P_1 \& P_2 := \text{expr}(P_1) \& \text{expr}(P_2)$. Additional properties, such as associativity of $\&$, can be enabled via additional trivial identities. Like us, their \ominus operator ensures that complemented expressions generate deterministic automata.

For basic (weighted) expressions, completely different approaches build the derived-term automaton with a quadratic complexity [1, 7]. However, the expansion-based algorithm features some unique properties. It supports a simple and natural on-the-fly implementation. It provides insight on the built automata by labeling states with the language/series they denote (e.g., Vcsn renders derived-term automata as in Ex. 19 and 21 to 23). It is a flexible framework in which new operators can be easily supported (e.g., the shuffle and infiltration operators in Vcsn). It supports the direct construction of deterministic automata. And it copes easily with alternative derivation schemes, such as the “broken derived-terms” [10, 11, 12, 2].

6 Conclusion

The construction of the derived-term automaton from a weighted rational expression is a powerful technique: states have a natural interpretation (they are identified by their future: the series they compute), extended rational expressions are easily supported, determinism can be requested, and it even offers a natural lazy, on-the-fly, implementation to handle infinite automata.

To build the derived-term automaton, we generalized Brzozowski's expansions to weighted expressions, and an inductive algorithm to compute the expansion of a rational expression. The formulas on which this algorithm is built reunite as a unique entity three facets that were kept separated in previous works: constant term, firsts, and derivatives. This results in a simpler set of equations, and an implementation whose complexity is independent of the size of the alphabet and even applies when it is infinite (e.g., when labels are strings, integers, etc.). Building the derived-term automaton using expansions is straightforward. Derivatives are only a technical tool to prove the correctness of the derived-terms. We have also shown that using proper techniques, the complexity of the algorithm is much better than previously reported.

The computation of expansions and derivations are implemented in `Vcsn`¹, together with their automaton construction procedures (possibly lazy, possibly deterministic). Our implementation actually prototypes support for additional operators on rational expressions (e.g., shuffle and infiltration). Our future work is focused on these operators.

Acknowledgments Interactions with A. Duret-Lutz, S. Lombardy, L. Saiu and J. Sakarovitch resulted in this work. Anonymous reviewers made very helpful comments.

References

- 1 C. Allauzen and M. Mohri. A unified construction of the Glushkov, follow, and Antimirov automata. In *MFCS*, vol. 4162 of *LNCS*, pp. 110–121. Springer, 2006.
- 2 P.-Y. Angrand, S. Lombardy, and J. Sakarovitch. On the number of broken derived terms of a rational expression. *Journal of Automata, Languages and Combinatorics*, 15(1/2): 27–51, 2010.
- 3 V. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *TCS*, 155(2):291–319, 1996.
- 4 J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.
- 5 P. Caron, J.-M. Champarnaud, and L. Mignot. Partial derivatives of an extended regular expression. In *LATA*, vol. 6638 of *LNCS*, pp. 179–191. Springer, 2011.
- 6 J.-M. Champarnaud and D. Ziadi. Canonical derivatives, partial derivatives and finite automaton constructions. *TCS*, 289(1):137–163, 2002.
- 7 J.-M. Champarnaud, F. Ouardi, and D. Ziadi. An efficient computation of the equation \mathbb{K} -automaton of a regular \mathbb{K} -expression. In *DLT*, vol. 4588 of *LNCS*. Springer, 2007.
- 8 A. Demaille, A. Duret-Lutz, S. Lombardy, and J. Sakarovitch. Implementation concepts in Vaucanson 2. In *CIAA'13*, vol. 7982 of *LNCS*, pp. 122–133, July 2013. Springer.
- 9 V. M. Glushkov. The abstract theory of automata. *Russian Math. Surveys*, 16:1–53, 1961.
- 10 S. Lombardy and J. Sakarovitch. How expressions can code for automata. In *LATIN*, pp. 242–251, 2004.

- 11 S. Lombardy and J. Sakarovitch. Derivatives of rational expressions with multiplicity. *TCS*, 332(1-3):141–177, 2005.
- 12 S. Lombardy and J. Sakarovitch. Corrigendum to our paper: How expressions can code for automata. *RAIRO — Theoretical Informatics and Applications*, 44(3):339–361, 2010.
- 13 R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9:39–47, 1960.
- 14 S. Owens, J. Reppy, and A. Turon. Regular-expression derivatives re-examined. *J. Funct. Program.*, 19(2):173–190, Mar. 2009.
- 15 J. J. M. M. Rutten. Automata, power series, and coinduction: Taking input derivatives seriously. In *Automata, Languages and Programming, 26th International Colloquium, ICALP’99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, vol. 1644 of *LNCS*, pp. 645–654. Springer, 1999.
- 16 J. J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *TCS*, 308(1-3):1–53, 2003.
- 17 J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009. Corrected English translation of *Éléments de théorie des automates*, Vuibert, 2003.

A

 Appendix

Proof of Lemma 12. Most operators are trivial, we focus here on the extended operators.

$$\begin{aligned}
 \llbracket X \& Y \rrbracket &= \left\llbracket \langle X_\varepsilon Y_\varepsilon \rangle \oplus \bigoplus_{a \in f(X) \cap f(Y)} a \odot \llbracket X_a \& Y_a \rrbracket \right\llbracket && \text{by definition, (8)} \\
 &= X_\varepsilon Y_\varepsilon + \sum_{a \in f(X) \cap f(Y)} a \cdot \llbracket X_a \& Y_a \rrbracket && \text{by definition of expr} \\
 &= X_\varepsilon Y_\varepsilon + \sum_{a \in f(X) \cap f(Y)} a \cdot (\llbracket X_a \rrbracket \& \llbracket Y_a \rrbracket) && \text{by Lemma 9} \\
 &= \left(X_\varepsilon + \sum_{a \in f(X)} a \cdot \llbracket X_a \rrbracket \right) \& \left(Y_\varepsilon + \sum_{a \in f(Y)} a \cdot \llbracket Y_a \rrbracket \right) && \text{by (2)} \\
 &= \left\llbracket \langle X_\varepsilon \rangle \oplus \bigoplus_{a \in f(X)} a \cdot \llbracket X_a \rrbracket \right\llbracket \& \left\llbracket \langle Y_\varepsilon \rangle \oplus \bigoplus_{a \in f(Y)} a \cdot \llbracket Y_a \rrbracket \right\llbracket \\
 &= \llbracket X \rrbracket \& \llbracket Y \rrbracket
 \end{aligned}$$

$$\begin{aligned}
 \llbracket X^c \rrbracket &= \left\llbracket \langle X_\varepsilon^c \rangle \oplus \bigoplus_{a \in f(X)} a \odot \llbracket X_a^c \rrbracket \oplus \bigoplus_{a \in A \setminus f(X)} a \odot \llbracket 0^c \rrbracket \right\llbracket && \text{by definition, (9)} \\
 &= X_\varepsilon^c + \sum_{a \in f(X)} a \cdot \llbracket X_a^c \rrbracket + \sum_{a \in A \setminus f(X)} a \cdot \llbracket 0^c \rrbracket \\
 &= X_\varepsilon^c + \sum_{a \in f(X)} a \cdot \llbracket X_a \rrbracket^c + \sum_{a \in A \setminus f(X)} a \cdot \llbracket 0 \rrbracket^c && \text{by Lemma 9} \\
 &= \left(X_\varepsilon + \sum_{a \in f(X)} a \cdot \llbracket X_a \rrbracket \right)^c && \text{by (3)}
 \end{aligned}$$

$$= \llbracket X \rrbracket^c$$

► **Example 24** (Ex. 23 detailed). *We have:*

$$d((ab)^c) = d(ab)^c = (\langle a \rangle \odot [b])^c = \langle 1_{\mathbb{K}} \rangle \oplus a \odot [b^c] \oplus b \odot [0^c]$$

$$\begin{aligned} d(\langle 3 \rangle (a+b)(a+b)^*) &= \langle 3 \rangle d((a+b)(a+b)^*) \\ &= \langle 3 \rangle d_p(a+b) \cdot (a+b)^* \oplus \langle d_\varepsilon(a+b) \rangle d((a+b)^*) \\ &= \langle 3 \rangle (a \odot [\langle 1 \rangle \odot 1] \oplus b \odot [\langle 1 \rangle \odot 1]) \cdot (a+b)^* \oplus \langle 0_{\mathbb{K}} \rangle d((a+b)^*) \\ &= \langle 3 \rangle (a \odot [\langle 1 \rangle \odot (a+b)^*] \oplus b \odot [\langle 1 \rangle \odot (a+b)^*]) \\ &= a \odot [\langle 3 \rangle \odot (a+b)^*] \oplus b \odot [\langle 3 \rangle \odot (a+b)^*] \end{aligned}$$

therefore:

$$\begin{aligned} X &:= d((ab)^c) \& d(\langle 3 \rangle (a+b)(a+b)^*) \\ &= (\langle 1_{\mathbb{K}} \rangle \oplus a \odot [b^c] \oplus b \odot [0^c]) \& (\langle a \rangle \odot [\langle 3 \rangle \odot (a+b)^*] \oplus \langle b \rangle \odot [\langle 3 \rangle \odot (a+b)^*]) \\ &= a \odot [\langle 3 \rangle \odot (b^c \& (a+b)^*)] \oplus b \odot [\langle 3 \rangle \odot (0^c \& (a+b)^*)] \\ &= a \odot [\langle 3 \rangle \odot (b^c \& (a+b)^*)] \oplus b \odot [\langle 3 \rangle \odot (a+b)^*] \end{aligned}$$

and finally

$$\begin{aligned} d(E_3) &= d(\langle 2 \rangle ab + (ab)^c \& \langle 3 \rangle (a+b)(a+b)^*) \\ &= d(\langle 2 \rangle ab) \oplus d((ab)^c \& \langle 3 \rangle (a+b)(a+b)^*) \\ &= a \odot [\langle 2 \rangle \odot b] \oplus \overbrace{d((ab)^c) \& d(\langle 3 \rangle (a+b)(a+b)^*)}^{\times} \\ &= a \odot [\langle 2 \rangle \odot b] \oplus a \odot [\langle 3 \rangle \odot (b^c \& (a+b)^*)] \oplus b \odot [\langle 3 \rangle \odot (a+b)^*] \\ &= a \odot [\langle 2 \rangle \odot b \oplus \langle 3 \rangle \odot (b^c \& (a+b)^*)] \oplus b \odot [\langle 3 \rangle \odot (a+b)^*] \end{aligned}$$

B Appendix: Proof of Theorem 20

Proving this theorem requires several auxiliary results. None of them is needed in an implementation: Def. 14 is all that is needed to build the derived-term automaton.

The path, paved by Lombardy and Sakarovitch [11], is as follows. First, define derivation with respect to a word, and show that it is a syntactic “implementation” of left-quotient of a series by a word (Appendix B.1). Then define (syntactically) the set of derived terms, and show that they generate all the word derivatives (Appendix B.2). Finally show that computations in the derived-term automaton correspond to computing the left-quotient of the denoted series (Appendix B.3).

This is also the path followed by the rather terse proof of Caron et al. [5, Proposition 4], but filling the gaps.

B.1 Derivation by Words

► **Definition 25** (Derivation of a Polynomial). $\partial_a \oplus_{i \in I} \langle k_i \rangle \odot E_i := \oplus_{i \in I} \langle k_i \rangle \odot \partial_a E_i$

► **Lemma 26.**

$$\partial_a(P \& Q) = \partial_a P \& \partial_a Q \tag{26}$$

$$\partial_a \text{expr}(P) = \text{expr}(\partial_a P) \tag{27}$$

$$\partial_a(P^c) = (\partial_a P)^c \tag{28}$$

Proof. Let $P := \bigoplus_{i \in I} \langle k_i \rangle \odot E_i$, $Q := \bigoplus_{j \in J} \langle h_j \rangle \odot F_j$.

$$\begin{aligned}
 \partial_a(P \& Q) &= \partial_a \left(\bigoplus_{i \in I, j \in J} \langle k_i h_j \rangle \odot E_i \& F_j \right) && \text{by def. of polynomial conjunction} \\
 &= \bigoplus_{i \in I, j \in J} \langle k_i h_j \rangle \odot \partial_a(E_i \& F_j) \\
 &= \bigoplus_{i \in I, j \in J} \langle k_i h_j \rangle \odot (\partial_a E_i \& \partial_a F_j) && \text{by (23)} \\
 &= \partial_a P \& \partial_a Q && \text{by def. of polynomial conjunction}
 \end{aligned}$$

$$\begin{aligned}
 \partial_a \text{expr}(P) &= \partial_a \text{expr} \left(\bigoplus_{i \in I} \langle k_i \rangle \odot E_i \right) \\
 &= \partial_a \sum_{i \in I} \langle k_i \rangle E_i \\
 &= \sum_{i \in I} \langle k_i \rangle \partial_a E_i \\
 &= \text{expr} \left(\bigoplus_{i \in I} \langle k_i \rangle \odot \partial_a E_i \right) \\
 &= \text{expr}(\partial_a P)
 \end{aligned}$$

$$\begin{aligned}
 \partial_a(P^c) &= \partial_a(\text{expr}(P)^c) && \text{by def. of polynomial complement} \\
 &= \left(\partial_a(\text{expr}(P)) \right)^c && \text{by (24)} \\
 &= (\text{expr}(\partial_a P))^c && \text{by (27)} \\
 &= (\partial_a P)^c && \text{by def. of polynomial complement} \quad \blacktriangleleft
 \end{aligned}$$

Derivation wrt a single-letter word is defined as the derivation wrt that letter. Derivation wrt to a longer word is the result of repeated derivations wrt letters.

► **Definition 27** (Derivation wrt a Word). $\forall a \in A, u \in A^+, \partial_{ua}E := \partial_a \partial_u E$.

► **Lemma 28.** $\partial_{uv}E = \partial_v \partial_u E$

Explicit formulas exist for derivation with respect to a word.

► **Lemma 29** (Direct Computations of Derivation wrt a Word).

$$\partial_u(E + F) = \partial_u E \oplus \partial_u F, \quad (29)$$

$$\partial_u(\langle k \rangle E) = \langle k \rangle (\partial_u E), \quad (30)$$

$$\partial_u(E \langle k \rangle) = (\partial_u E) \langle k \rangle, \quad (31)$$

$$\partial_u(E \cdot F) = (\partial_u E) \cdot F \oplus \left(\bigoplus_{\substack{f=gh \\ g \in A^*, h \in A^+}} c(\partial_g E) \partial_h F \right) \quad (32)$$

$$\partial_u E^* = \bigoplus_{\substack{f=g_1 g_2 \dots g_n \\ g_1, \dots, g_n \in A^+}} \left\langle \left(\prod_{i \in [n-1]} c(E)^* c(\partial_{g_i} E) \right) c(E)^* \right\rangle \partial_{g_n} E \cdot E^* \quad (33)$$

$$\partial_u (E \& F) = \partial_u E \& \partial_u F, \quad (34)$$

$$\partial_u E^c = (\partial_u E)^c \quad (35)$$

Proof. The proof is the same as that of [11, Prop. 3], with additional cases for conjunction and complement.

For conjunction:

$$\begin{aligned} \partial_{ua} (E \& F) &= \partial_a \partial_u (E \& F) \\ &= \partial_a (\partial_u E \& \partial_u F) && \text{by induction hypothesis} \\ &= \partial_a \partial_u E \& \partial_a \partial_u F && \text{by (26)} \\ &= \partial_{ua} E \& \partial_{ua} F \end{aligned}$$

For complement:

$$\begin{aligned} \partial_{ua} (E^c) &= \partial_a \partial_u (E^c) \\ &= \partial_a (\partial_u E)^c && \text{by induction hypothesis} \\ &= (\partial_a (\partial_u E))^c && \text{by (28)} \\ &= (\partial_{ua} E)^c \end{aligned} \quad \blacktriangleleft$$

The following lemma makes explicit the connection between the (syntactic) derivation, and the semantics of an expression.

► **Lemma 30** ([11, Prop. 4]). $\forall u \in A^+, \llbracket E \rrbracket(u) = c(\partial_u E)$.

Proof. For conjunction:

$$\begin{aligned} \llbracket E \& F \rrbracket(u) &= (\llbracket E \rrbracket \& \llbracket F \rrbracket)(u) && \text{by definition} \\ &= \llbracket E \rrbracket(u) \cdot \llbracket F \rrbracket(u) && \text{by definition} \\ &= c(\partial_u E) \cdot c(\partial_u F) && \text{by induction hypothesis} \\ &= c(\partial_u E \& \partial_u F) && \text{by (23)} \\ &= c(\partial_u (E \& F)) && \text{by (34)} \end{aligned}$$

For complement:

$$\begin{aligned} \llbracket E^c \rrbracket(u) &= \llbracket E \rrbracket^c(u) && \text{by definition} \\ &= (\llbracket E \rrbracket(u))^c && \text{by definition} \\ &= (c(\partial_u E))^c && \text{by induction hypothesis} \\ &= c((\partial_u E)^c) && \text{by (24)} \\ &= c(\partial_u (E^c)) && \text{by (35)} \end{aligned} \quad \blacktriangleleft$$

The previous lemma allows to show the connection between the (syntactic) derivation, and the (semantical) left-quotient of a series.

► **Theorem 31** ([11, Theorem 1]). $\forall u \in A^+, \llbracket \partial_u E \rrbracket = u^{-1} \llbracket E \rrbracket$.

Proof. For any word $v \in A^+$,

$$\begin{aligned}
 \llbracket \partial_u E \rrbracket(v) &= c(\partial_v \partial_u E) && \text{by Lemma 30} \\
 &= c(\partial_{uv} E) && \text{by Lemma 28} \\
 &= \llbracket E \rrbracket(uv) && \text{by Lemma 30} \\
 &= (u^{-1} \llbracket E \rrbracket)(v) && \text{by definition of left-quotient}
 \end{aligned}$$

B.2 Derived Terms

► **Definition 32** (Derived Terms). *Given an expression E , its derived terms is the set $D(E)$ defined as follows:*

$$\begin{aligned}
 D(0) &:= \emptyset \\
 D(1) &:= \emptyset \\
 D(a) &:= \{1\} \quad \forall a \in A \\
 D(E + F) &:= D(E) \cup D(F) \\
 D(\langle k \rangle E) &:= D(E) \quad \forall k \in \mathbb{K} \\
 D(E \langle k \rangle) &:= \{E_i \langle k \rangle \mid E_i \in D(E)\} \quad \forall k \in \mathbb{K} \\
 D(E \cdot F) &:= \{E_i \cdot F \mid E_i \in D(E)\} \cup D(F) \\
 D(E^*) &:= \{E_i \cdot E^* \mid E_i \in D(E)\} \\
 D(E \& F) &:= \{E_i \& F_j \mid \forall E_i \in D(E), \forall F_j \in D(F)\} \\
 D(E^c) &:= \{(\langle k_1 \rangle E_1 + \dots + \langle k_n \rangle E_n)^c \mid \forall k_1, \dots, k_n \in \mathbb{K}, \forall E_1, \dots, E_n \in D(E)\}
 \end{aligned}$$

where in the last equation, the E_i are sorted. Besides, depending on the features of \mathbb{K} , the coefficients may be normalized so that colinear combinations are represented only once. For instance if \mathbb{K} has no zero divisor, one may divide by the GCD of the k_i (so that the k_i are coprime), or, in the case of a field, by the first non null k_i (so that the first non null coefficient is $1_{\mathbb{K}}$), or by the sum of the k_i provided it's not null (so that the sum of the coefficients is $1_{\mathbb{K}}$), etc.

► **Theorem 33.** *If \mathbb{K} is finite, or if E has no complement, then $D(E)$ is finite.*

Proof. This is a direct consequence from Def. 32: finiteness propagates during the induction. The only danger is the case of complement, whose finiteness ensues from a very crude criterion: there exists a finite number of combinations. ◀

We prove that the set of derived terms is closed by derivation. The insightful reader can see automata dawning: the derived terms are the states, and the coefficients are the weights of the transitions.

► **Lemma 34.** *We denote $\{1, \dots, n\}$ by $[n]$.*

Let E be an expression, $D(E) = \{E_i \mid i \in [n]\}$ be its derived terms. There exists n coefficients $(k_i^{(a)})_{i \in [n]}$ and n^2 coefficients $(k_{i,j}^{(a)})_{i,j \in [n]}$ such that

$$\partial_a E = \bigoplus_{i \in [n]} \langle k_i^{(a)} \rangle E_i \quad \partial_a E_i = \bigoplus_{i' \in [n]} \langle k_{i,i'}^{(a)} \rangle E_{i'}$$

Proof. We follow [11, proof of Theorem 2], to which we add the following cases. We note:

$$D(F) = \{F_j \mid j \in [m]\} \quad \partial_a F = \bigoplus_{j \in [m]} \langle h_j^{(a)} \rangle F_j \quad \partial_a F_j = \bigoplus_{j' \in [m]} \langle h_{j,j'}^{(a)} \rangle F_{j'}$$

Consider $E \& F$:

$$\begin{aligned}\partial_a(E \& F) &= \partial_a E \& \partial_a F \\ &= \left(\bigoplus_{i \in [n]} \langle k_i^{(a)} \rangle E_i \right) \& \left(\bigoplus_{j \in [m]} \langle h_j^{(a)} \rangle F_j \right) \\ &= \bigoplus_{i \in [n], j \in [m]} \langle k_i^{(a)} h_j^{(a)} \rangle (E_i \& F_j)\end{aligned}$$

which is indeed a linear combination of derived terms of $E \& F$, since $D(E \& F) = \{E_i \& F_j \mid \forall E_i \in D(E), \forall F_j \in D(F)\}$ by definition Def. 32.

Likewise,

$$\begin{aligned}\partial_a(E_i \& F_j) &= \partial_a E_i \& \partial_a F_j \\ &= \left(\bigoplus_{i' \in [n]} \langle k_{i,i'}^{(a)} \rangle E_{i'} \right) \& \left(\bigoplus_{j' \in [m]} \langle h_{j,j'}^{(a)} \rangle F_{j'} \right) \\ &= \bigoplus_{i' \in [n], j' \in [m]} \langle k_{i,i'}^{(a)} h_{j,j'}^{(a)} \rangle (E_{i'} \& F_{j'})\end{aligned}$$

is a linear combination of elements of $D(E \& F)$.

Consider E^c :

$$\begin{aligned}\partial_a(E^c) &= (\partial_a E)^c \\ &= \left(\bigoplus_{i \in [n]} \langle k_i^{(a)} \rangle E_i \right)^c \\ &= \left(\text{expr} \left(\bigoplus_{i \in [n]} \langle k_i^{(a)} \rangle E_i \right) \right)^c \\ &= \left(\sum_{i \in [n]} \langle k_i^{(a)} \rangle E_i \right)^c\end{aligned}$$

which is a member of $D(E^c)$. Note in this case, we expect the E_i to be sorted in the same order as the one used by `expr`.

Besides:

$$\begin{aligned}\partial_a \left(\left(\sum_{i \in [n]} \langle k_i^{(a)} \rangle E_i \right)^c \right) &= \left(\partial_a \left(\sum_{i \in [n]} \langle k_i^{(a)} \rangle E_i \right) \right)^c \\ &= \left(\bigoplus_{i \in [n]} \langle k_i^{(a)} \rangle \partial_a E_i \right)^c \\ &= \left(\bigoplus_{i \in [n]} \langle k_i^{(a)} \rangle \bigoplus_{i' \in [n]} \langle k_{i,i'}^{(a)} \rangle E_{i'} \right)^c \\ &= \left(\bigoplus_{i, i' \in [n]} \langle k_i^{(a)} k_{i,i'}^{(a)} \rangle E_{i'} \right)^c\end{aligned}$$

$$= \left(\sum_{i, i' \in [n]} \langle k_i^{(a)} k_{i'}^{(a)} \rangle E_{i'} \right)^c$$

which is a member of $D(E^c)$. ◀

The following result, similar to [11, Theorem 3], shows that any word derivative of an expression is a linear combination of its derived terms.

► **Theorem 35.** *Let E be an expression, $D(E) = \{E_i \mid i \in [n]\}$ be its derived terms, and $u \in A^+$ any word. There exist coefficients $(k_i^{(u)})_{i \in [n]}$ in \mathbb{K} such that:*

$$\partial_u E = \bigoplus_{i \in [n]} \langle k_i^{(u)} \rangle E_i$$

Proof. The result is proved by induction.

The base case is established by Lemma 34.

$$\partial_{ua} E = \partial_a \partial_u E$$

$$= \partial_a \left(\bigoplus_{i \in [n]} \langle k_i^{(u)} \rangle E_i \right) \quad \text{by induction hypothesis}$$

$$= \bigoplus_{i \in [n]} \langle k_i^{(u)} \rangle \partial_a E_i$$

$$= \bigoplus_{i \in [n]} \langle k_i^{(u)} \rangle \left(\bigoplus_{j \in [n]} \langle k_{i,j}^{(a)} \rangle E_j \right) \quad \text{by Lemma 34}$$

$$= \bigoplus_{j \in [n]} \left(\bigoplus_{i \in [n]} \langle k_i^{(u)} k_{i,j}^{(a)} \rangle E_j \right)$$

$$= \bigoplus_{j \in [n]} \left\langle \sum_{i \in [n]} k_i^{(u)} k_{i,j}^{(a)} \right\rangle E_j$$

i.e.,

$$k_j^{(ua)} = \sum_{i \in [n]} k_i^{(u)} k_{i,j}^{(a)} \quad (36)$$

◀

B.3 Derived-term Automaton

In order to prove the final result, we express automata in a different way [11, Sect. 5].

► **Definition 36** (Representations of a Finite Weighted Automaton). *The matrix representation of a (finite weighted) automaton is the sextuplet $\langle A, \mathbb{K}, E, Q, E, I, T \rangle$ where:*

- A is an alphabet
- \mathbb{K} (the set of weights) is a semiring,
- Q is a finite set of states,
- I (resp. T) is a row (resp. column) vector of dimension Q with entries in \mathbb{K} ,
- E is a square matrix whose entries are linear combinations of letters of A with coefficients in \mathbb{K} .

The \mathbb{K} -representation of an automaton is the triple $\langle I, \zeta, T \rangle$ where ζ is a morphism from A to $\mathbb{K}^{Q \times Q}$ such that $E = \sum_{a \in A} \zeta(a)a$.

One can then prove that, for every word $u \in A^*$:

$$\llbracket \mathcal{A} \rrbracket(u) = (I \cdot E^* \cdot T)(u) = (I \cdot E^{|u|} \cdot T)(u) = I \cdot \zeta(u) \cdot T$$

Put together, the definition of derivation and constant terms (Def. 16), their connection with expansions (Prop. 17), the definition of \mathcal{A}_E , the expansion-based derived-term automaton of E (Def. 18), and finally Lemma 34, show that \mathcal{A}_E admits the following \mathbb{K} -representation:

$$I_{E_i} = \begin{cases} 1_{\mathbb{K}} & \text{if } E_i = E \\ 0_{\mathbb{K}} & \text{otherwise} \end{cases} \quad \zeta(a)_{i,j} = k_{i,j}^{(a)} \quad T_{E_i} = c(E_i)$$

where the coefficients $k_{i,j}^{(a)}$ were defined in Lemma 34. The E_i are the derived-terms of E , to which we add $E_0 := E$ if $E \notin D(E)$, in which case $k_{i,0}^{(a)} := 0_{\mathbb{K}}$, and $k_{0,i}^{(a)} := k_i^{(a)}$ for all $i > 0$.

We prove by induction that:

$$\forall u \in A^+, \forall i \in [n], (I \cdot \zeta(u))_i = k_i^{(u)} \quad (37)$$

Proof. The base case:

$$\begin{aligned} (I \cdot \zeta(a))_i &= \sum_j (I_j \cdot \zeta(a)_{j,i}) \\ &= 1_{\mathbb{K}} \cdot \zeta(a)_{0,i} && \text{by definition of } I \\ &= k_{0,i}^{(a)} && \text{by definition of } \zeta \\ &= k_i^{(a)} && \text{by definition of } k_{i,j}^{(a)} \end{aligned}$$

Then the induction:

$$\begin{aligned} (I \cdot \zeta(ua))_i &= (I \cdot (\zeta(u) \cdot \zeta(a)))_i \\ &= ((I \cdot \zeta(u)) \cdot \zeta(a))_i \\ &= \sum_j (I \cdot \zeta(u))_j \cdot \zeta(a)_{j,i} \\ &= \sum_j (k_j^{(u)} \cdot \zeta(a)_{j,i}) && \text{by induction hypothesis} \\ &= \sum_j (k_j^{(u)} \cdot k_{j,i}^{(a)}) && \text{by definition of } \zeta \\ &= k_i^{(ua)} && \text{by (36)} \end{aligned} \quad \blacktriangleleft$$

We can now finally prove that $\llbracket \mathcal{A}_E \rrbracket = \llbracket E \rrbracket$. Let $u \in A^+$:

$$\begin{aligned} \llbracket \mathcal{A}_E \rrbracket(u) &= (I \cdot \zeta(u) \cdot T) \\ &= \sum_i (I \cdot \zeta(u))_i \cdot T_i \\ &= \sum_i k_i^{(u)} \cdot T_i && \text{by (37)} \\ &= \sum_i k_i^{(u)} \cdot c(E_i) && \text{by definition of } T \end{aligned}$$

$$\begin{aligned}
&= c \left(\bigoplus_i \langle k_i^{(u)} \rangle E_i \right) \\
&= c(\partial_u E) && \text{by Theorem 35} \\
&= \llbracket E \rrbracket(u) && \text{by Theorem 31}
\end{aligned}$$

The case of the empty word follows from the definition of I and T : $\llbracket \mathcal{A}_E \rrbracket(\varepsilon) = \sum_i (I_i \cdot T_i) = 1_{\mathbb{K}} \cdot T_0 = c(E)$.